

Examen pratique de janvier 2024

JEUX DE SOCIÉTÉ

Une communauté d'amateurs de jeux de société souhaite disposer d'une application leur permettant de déterminer le **nombre de joueurs idéal** pour un jouer à un jeu de société spécifique.

Prenons par exemple le jeu « *Trio* » qui se joue de 3 à 6 joueurs. Chaque membre de la communauté peut partager son expérience de jeu en donnant une appréciation pour chaque nombre de joueurs (à 3, à 4, à 5 et à 6 dans le cas du jeu « *Trio* »). Les **appréciations** sont les suivantes : « *Meilleur* », « *Recommandé* », « *Pas recommandé* ».

Il en résulte un tableau récapitulatif permettant de conseiller les personnes intéressées par ce jeu (voir la figure 1 ci-dessous).

# JOUEURS	MEILLEUR	RECOMMANDÉ	PAS RECOMMANDÉ	# VOTES
3	15,0% (3)	60,0% (12)	25,0% (5)	20
4	77,8% (21)	22,2% (6)	0,0% (0)	27
5	27,8% (5)	72,2% (13)	0,0% (0)	18
6	7,7% (1)	61,5% (8)	30,8% (4)	13

Fig. 1 – Tableau répertoriant les avis des joueurs pour le jeu « *Trio* ».

Il ressort de ces résultats que le nombre de joueurs idéal pour le jeu « *Trio* » est de 4.

Ces chiffres sont obtenus sur base du vote de 31 membres, tous ne s'étant pas nécessairement prononcés pour chaque nombre de joueurs. En effet, un membre a la possibilité de ne pas se prononcer pour un nombre de joueurs, grâce à l'**option « NSP »** (qui signifie « *Ne se prononce pas* »), s'il n'a pu expérimenter le jeu dans cette configuration.

Pour mieux comprendre ce tableau, considérons sa première ligne correspondant à 3 joueurs. Sur les 31 votants, seuls 20 membres (3 + 12 + 5) se sont prononcés pour ce nombre de joueurs :

- 3 l'ont considéré comme le meilleur ($3 / 20 = 15,0\%$) ;
- 12 l'ont recommandé ($12 / 20 = 60,0\%$) ;
- 5 ne l'ont pas recommandé ($5 / 20 = 25,0\%$) ;
- et, par conséquent, 11 ne se sont pas prononcés.

*Notez que les pourcentages sont calculés **en fonction du nombre de membres qui se sont prononcés** et non du nombre total de votants. Par exemple, le pourcentage des membres qui considèrent que « 3 joueurs » est le meilleur nombre est de 15,0% ($= 3 / 20$) et non de 9,7% ($= 3 / 31$).*

DESCRIPTION DE L'APPLICATION

Au lancement de l'application, l'utilisateur doit encoder le **nombre minimum** et le **nombre maximum de joueurs** admis pour le jeu.

Un **menu** est ensuite affiché, proposant 3 options à l'utilisateur : « Voter », « Consulter les résultats » et « Quitter ».

La 1^{ère} option, « **Voter** », permet à l'utilisateur de donner son appréciation pour chaque nombre de joueurs.

La 2^e option, « **Consulter les résultats** », permet à l'utilisateur de consulter le tableau récapitulatif des votes.

*Référez-vous aux **exemples d'exécution** donnés en fin d'énoncé afin de mieux comprendre le résultat attendu.*

AVANT-PROPOS

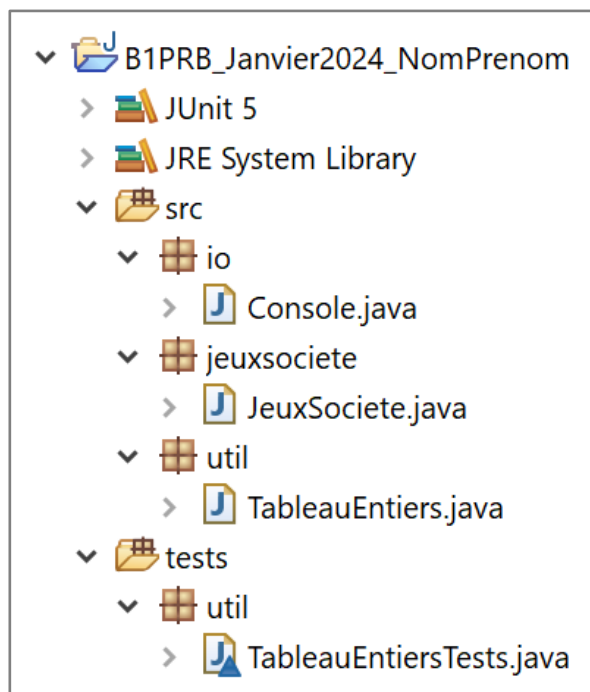
À deux exceptions près, les types **String** et **PrintStream**, vous ne pouvez pas utiliser le paradigme orienté objet pour coder cette application. N'oubliez pas de qualifier toutes vos fonctions de **static** !

PRÉPARATION

1. Dans Eclipse, créez un projet Java nommé **B1PRB_Janvier2024_NomPrenom**, où **NomPrenom** doit être remplacé par vos nom et prénom.

N'oubliez pas de vérifier que l'encodage utilisé pour les fichiers est l'**UTF-8**. Pour ce faire, il faut suivre le chemin : *Window > Preferences > General > Workspace > Text file encoding*.

2. Reproduisez exactement l'organisation suivante :



La classe **Console** est à télécharger au début de la page *HELMo Learn* du cours. Les autres classes doivent être créées par vos soins. La fonction **main** doit être déclarée dans la classe **JeuxSociete**.

*Le projet déposé sur l'espace de cours doit être **complet et exécutable** !*

FONCTIONNALITÉS INDISPENSABLES


Les fonctionnalités suivantes doivent être opérationnelles, tout en veillant à respecter les consignes données dans leur description :

- La fonction *sommer* de la classe **TableauEntiers** doit être correctement réalisée, testée et utilisée.
- La fonction *afficherTableauRecapitulatif* de la classe **JeuxSociete** doit être correctement réalisée, *au moins dans sa version minimale*, et utilisée.
- Dans la fonction *main* de la classe **JeuxSociete**, un tableau de chaînes de caractères à une dimension doit être correctement créé (voir la section « Les structures de données ») et initialisé avec les **appellations des trois appréciations** : « *Meilleur* », « *Recommandé* », « *Pas recommandé* ».
- Dans la fonction *main* de la classe **JeuxSociete**, un tableau d'entiers à deux dimensions doit être correctement créé et initialisé afin d'y **comptabiliser les votes** (voir la section « Les structures de données »).
- La fonction *main* doit permettre l'acquisition des nombres de joueurs minimum et maximum, l'acquisition des appréciations d'au moins un membre, puis l'affichage du tableau récapitulatif (voir l'exemple d'exécution « MINIMUM REQUIS » en fin d'énoncé).

Dans la version minimale, il n'est pas nécessaire de vérifier la validité des données saisies par l'utilisateur.

Si l'une de ces fonctionnalités n'est pas observée, cela entraîne automatiquement l'échec pour l'examen. Attention, la réussite de ces fonctionnalités seules ne garantit pas l'obtention d'une note supérieure ou égale à 10 / 20 ! Ne vous contentez pas de ces fonctionnalités et veillez à soigner les tests unitaires et la javadoc des fonctions.

APPROCHE RECOMMANDÉE

Dans un premier temps, **ne réalisez que les fonctions indispensables**. Ces dernières sont indiquées dans la section « *Fonctionnalités indispensables* » et sont marquées d'un point d'exclamation  au niveau de leurs descriptions.

Réalisez ensuite la fonction *main* sur base des directives données dans la section « *Fonctionnalités indispensables* » et de l'exemple d'exécution **MINIMUM REQUIS**.

Il est recommandé de ne réaliser la suite du programme que lorsque ces deux premières étapes sont terminées.

LES STRUCTURES DE DONNÉES

ENREGISTRER LES APPELLATIONS DES APPRÉCIATIONS

Utilisez un **tableau de chaînes de caractères à une dimension** pour enregistrer les appellations des trois appréciations, dans cet ordre : « *Meilleur* », « *Recommandé* », « *Pas recommandé* ».

Le tableau obtenu est le suivant :

"Meilleur"	"Recommandé"	"Pas recommandé"
0	1	2

*Il est normal que le terme « **NSP** » n'apparaisse pas ici étant donné le rôle particulier que tient ce dernier, c'est-à-dire celui de ne pas donner d'appréciation.*

COMPTABILISER LES VOTES

Utilisez un **tableau de chaînes d'entiers à deux dimensions** pour comptabiliser les votes par nombre de joueurs et par appréciation.

Ce tableau doit comporter **autant de lignes que de configurations de jeu existantes**. Par exemple, pour le jeu « *Trio* » qui se joue de 3 à 6 joueurs, il y a quatre configurations possibles (à 3, à 4, à 5 et à 6 joueurs), ce qui implique l'emploi d'un tableau de 4 lignes.

Le tableau doit disposer d'**une colonne par appréciation**, ce qui représente 3 colonnes : la première pour l'appréciation « *Meilleur* », la seconde pour « *Recommandé* » et la troisième pour « *Pas recommandé* ».

Vous remarquerez que l'ordre utilisé ci-dessus est (volontairement) identique à celui des appellations des appréciations dans le tableau à une dimension. Cette particularité s'avérera utile dans la réalisation du programme.

Initialement, toutes les valeurs du tableau sont à 0 :

0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0

Dans le cas de l'exemple donné à la figure 1, le tableau correspondant est le suivant :

0	3	12	5
1	21	6	0
2	5	13	0
3	1	8	4

Voici comment interpréter les 3 valeurs apparaissant à la première ligne du tableau ci-dessus : à 3 joueurs, il y a 3 votes pour l'appréciation « *Meilleur* », 12 pour « *Recommandé* » et 5 pour « *Pas recommandé* ».

LES CLASSES `TableauEntiers` ET `TableauEntiersTests`

Toutes les fonctions de la classe `TableauEntiers` doivent être documentées avec des commentaires javadoc et testées à l'aide de JUnit 5.

Dans la classe `TableauEntiers`, déclarez :



- Une fonction nommée `sommer` qui retourne la somme des valeurs présentes dans un tableau d'entiers à une dimension :

```
public static int sommer(int[] t)
```

Le paramètre `t` est le tableau contenant les valeurs à sommer.

Exemple : si `t` est le tableau `[3, 12, 5]`, alors la fonction retourne `20`.

[CAS PARTICULIER] Si le tableau `t` est vide, la fonction retourne 0.

[PRÉCONDITION] La référence réceptionnée par le paramètre `t` est supposée valide (ne vaut pas `null`).

- Une fonction nommée `calculerTaux` qui retourne un tableau dans lequel les valeurs provenant d'un autre tableau ont été exprimées en taux :

```
public static double[] calculerTaux(int[] t)
```

Le paramètre `t` est le tableau contenant les valeurs à exprimer en taux.

Pour effectuer sa tâche, cette fonction doit utiliser la fonction `sommer(int[])` déclarée précédemment !

Exemple : si `t` est le tableau `[3, 12, 5]`, alors le tableau retourné est `[15.0, 60.0, 25.0]`.

[CAS PARTICULIER] Si la somme de l'ensemble des valeurs du tableau est égale à 0, alors le tableau retourné ne doit être composé que de valeurs `Double.NaN`.

Exemple : si `t` est le tableau `[0, 0, 0]`, alors le tableau retourné est `[NaN, NaN, NaN]`.

`NaN`].

[SUGGESTION] Vous pouvez utiliser la fonction `fill` de la classe `Arrays`.

[PRÉCONDITION] La référence réceptionnée par le paramètre `t` est supposée valide (ne vaut pas `null`).

→ Une fonction nommée `calculerTaux` qui retourne un tableau dans lequel les valeurs provenant d'un autre tableau ont été exprimées en taux, les taux d'une même ligne étant établis uniquement sur base des valeurs apparaissant dans cette ligne :

```
public static double[][] calculerTaux(int[][] t)
```

Le paramètre `t` est le tableau contenant les valeurs à exprimer en taux.

Pour effectuer sa tâche, cette fonction doit utiliser la fonction `calculerTaux(int[])` déclarée précédemment !

Exemple : si `t` est le tableau `[[3, 12, 5], [24, 8, 0], [2, 9, 4], [0, 0, 0]]`, alors le tableau retourné est `[[15.0, 60.0, 25.0], [75.0, 25.0, 0.0], [13.333..., 60.0, 26.666...], [NaN, NaN, NaN]]`.

[CAS PARTICULIER] La fonction doit pouvoir gérer la présence de références `null` en tant que références de ligne.

Exemple : si `t` est le tableau `[[3, 12, 5], [24, 8, 0], null, [0, 0, 0]]`, alors le tableau retourné est `[[15.0, 60.0, 25.0], [75.0, 25.0, 0.0], null, [NaN, NaN, NaN]]`.

[PRÉCONDITIONS] Le tableau réceptionné par la fonction est supposé valide (références du tableau et de ses lignes différentes de `null`).

LA CLASSE JEUXOCIETE

À l'exception de la fonction *main*, toutes les fonctions de la classe **JeuxSociete** doivent être documentées avec des commentaires javadoc. Cependant, aucune des fonctions de cette classe ne doit être testée à l'aide de JUnit 5.

Dans la classe **JeuxSociete**, déclarez :



→ Une fonction nommée *afficherTableauRecapitulatif* qui affiche le résultat des votes en indiquant pour chaque configuration de joueurs et chaque appréciation le nombre de votes récoltés et le taux correspondant, ainsi que le total des votes :

```
public static void afficherTableauRecapitulatif(
    String[] appreciations,
    int[][] votes,
    int nbMinJoueurs)
```

Le paramètre *appreciation* est un tableau contenant les appellations des appréciations.

Le paramètre *votes* est un tableau contenant les votes par nombre de joueurs et par appréciation.

Le paramètre *nbMinJoueurs* indique le nombre minimum de joueurs pour lequel le jeu est prévu.

[VERSION MINIMALE]

*Pour effectuer sa tâche, cette fonction doit utiliser la fonction *sommer(int[])* de la classe **TableauEntiers** déclarée précédemment !*

*Exemple : si *appreciations* est le tableau ["Meilleur", "Recommandé", "Pas recommandé"], *votes* le tableau [[3, 12, 5], [21, 6, 0], [5, 13, 0], [0, 0, 0]] et *nbMinJoueurs* l'entier 3, alors l'exécution de la fonction produit l'affichage suivant en console :*

```
# joueurs Meilleur Recommandé Pas recommandé # votes
```

```

3 (3) (12) (5) 20
4 (21) (6) (0) 27
5 (5) (13) (0) 18
6 (0) (0) (0) 0

```

[VERSION IDÉALE]

Dans sa version idéale, cette fonction doit utiliser la fonction `calculerTaux(int[][])` de la classe `TableauEntiers` déclarée précédemment !

Exemple : pour les mêmes valeurs que dans l'exemple précédent, l'exécution de la fonction produit l'affichage suivant en console :

# joueurs	Meilleur	Recommandé	Pas recommandé	# votes
3	15,0% (3)	60,0% (12)	25,0% (5)	20
4	77,8% (21)	22,2% (6)	0,0% (0)	27
5	27,8% (5)	72,2% (13)	0,0% (0)	18
6	- (0)	- (0)	- (0)	0

[CAS PARTICULIER] Si un taux équivaut à `Double.NaN`, alors un trait d'union `" - "` est affiché à la place.

[SUGGESTION] Aidez-vous des fonctions `isNaN` de la classe `Double`, `format` de la classe `String` et `printf` de la classe `PrintStream`.

[PRÉCONDITIONS] Les paramètres réceptionnés par la fonction sont supposés valides (références des tableaux et de leurs lignes différentes de `null`).

→ Une fonction nommée `lireEntier` qui effectue l'acquisition sécurisée d'un entier :

```
public static int lireEntier(String question)
```

Le paramètre `question` est la chaîne de caractères à afficher en guise de question pour spécifier la valeur que doit saisir l'utilisateur.

*L'acquisition **doit être répétée** tant que la saisie ne correspond pas à un entier. Pour déterminer cela, utilisez une **expression régulière** qui vérifie si la chaîne est constituée d'au moins un chiffre. Le nombre entier peut être précédé d'un signe (le moins ou le*

plus).

Exemple : si *question* est la chaîne de caractères "Nombre minimum de joueurs ? ", alors l'exécution de la fonction pourrait produire l'affichage suivant en console :

```
Nombre minimum de joueurs ? deux ↵
Nombre minimum de joueurs ? 2 joueurs ↵
Nombre minimum de joueurs ? 2.0 ↵
Nombre minimum de joueurs ? +2 ↵
```

L'acquisition est ici répétée quatre fois car la première saisie ne contient pas de chiffre, la seconde contient des caractères alphabétiques et la troisième représente un nombre réel.

Dans ce cas, la fonction retourne l'entier 2.

[SUGGESTION] Aidez-vous des fonctions *trim* et *matches* de la classe **String**, ainsi que *parseInt* de la classe **Integer**.

[PRÉCONDITION] La chaîne *question* est supposée valide (ne vaut pas *null*).

→ Une fonction nommée *lireEntier* qui effectue l'acquisition sécurisée d'un entier :

```
public static int lireEntier(String question, int min)
```

Le paramètre *question* est la chaîne de caractères à afficher en guise de question pour spécifier la valeur que doit saisir l'utilisateur.

Le paramètre *min* indique le nombre entier minimum admis.

L'acquisition doit être répétée tant que la saisie ne correspond pas à un entier supérieur ou égal à min.

*Pour effectuer sa tâche, cette fonction doit utiliser la fonction *lireEntier(String)* déclarée précédemment !*

Exemple : si `question` est la chaîne de caractères "Nombre minimum de joueurs ? " et `min` l'entier 1, alors l'exécution de la fonction pourrait produire l'affichage suivant en console :

```
Nombre minimum de joueurs ? deux ↵
Nombre minimum de joueurs ? -2 ↵
Nombre minimum de joueurs ? ↵
Nombre minimum de joueurs ? 2↵
```

L'acquisition est ici répétée quatre fois car la première saisie ne contient pas de chiffre, la seconde est un entier inférieur à 1 et la troisième est une chaîne vide.

Dans ce cas, la fonction retourne l'entier 2.

[PRÉCONDITION] La chaîne `question` est supposée valide (ne vaut pas `null`).

→ Une fonction nommée `lireCaractere` qui effectue l'acquisition sécurisée d'un caractère sous sa forme majuscule :

```
public static char lireCaractere(String question,
                                String admis)
```

Le paramètre `question` est la chaîne de caractères à afficher en guise de question pour spécifier la valeur que doit saisir l'utilisateur.

Le paramètre `admis` est une chaîne spécifiant tous les caractères admis. *La casse n'est pas prise en compte* (par exemple, les caractères 'A' et 'a' sont considérés comme équivalents).

L'acquisition doit être répétée tant que la saisie ne correspond pas à l'un des caractères spécifiés dans `admis`, et ce, sans tenir compte de la casse.

Pour effectuer sa tâche, cette fonction doit utiliser la fonction `lireEntier(String)` déclarée précédemment !

Exemple : si `question` est la chaîne de caractères "... pour 3 joueur(s) ? " et `admis` la chaîne "MRPN", alors l'exécution de la fonction pourrait produire l'affichage

suivant en console :

```
... pour 3 joueur(s) ? A ↵
... pour 3 joueur(s) ? ↵
... pour 3 joueur(s) ? MRPN ↵
... pour 3 joueur(s) ? n ↵
```

L'acquisition est ici répétée quatre fois car la première lettre saisie ne correspond pas à l'une des quatre spécifiées, la seconde saisie est une chaîne vide et la troisième est une chaîne contenant plus d'un caractère.

Dans ce cas, la fonction retourne l'entier 'N'.

[CAS PARTICULIER] Si `admis` est une chaîne vide, alors n'importe quel caractère est accepté.

[SUGGESTION] Aidez-vous des fonctions `toUpperCase`, `length`, `indexOf` et `charAt` de la classe `String`.

[PRÉCONDITIONS] Les chaînes `question` et `admis` sont supposées valides (aucune référence `null`).

LES EXEMPLES D'EXÉCUTION

MINIMUM REQUIS

```
Nombre minimum de joueurs ? 2
Nombre maximum de joueurs ? 4

Votre appréciation ...
(M)eilleur
(R)ecommandé
(P)as recommandé
(N)SP
... pour 2 joueur(s) ? n
... pour 3 joueur(s) ? r
... pour 4 joueur(s) ? m

# joueurs Meilleur Recommandé Pas recommandé # votes
2 (0) (0) (0) 0
```

```
3 (0) (1) (0) 1
4 (1) (0) (0) 1
```

VERSION IDÉALE

```
Nombre minimum de joueurs ? 2
Nombre maximum de joueurs ? 4
```

```
MENU :
1. Voter
2. Consulter les résultats
3. Quitter
Choix ? 1
```

```
Votre appréciation ...
(M)eilleur
(R)ecommandé
(P)as recommandé
(N)SP
... pour 2 joueur(s) ? n
... pour 3 joueur(s) ? r
... pour 4 joueur(s) ? m
```

```
MENU :
1. Voter
2. Consulter les résultats
3. Quitter
Choix ? 1
```

```
Votre appréciation ...
(M)eilleur
(R)ecommandé
(P)as recommandé
(N)SP
... pour 2 joueur(s) ? n
... pour 3 joueur(s) ? m
... pour 4 joueur(s) ? n
```

```
MENU :
1. Voter
2. Consulter les résultats
3. Quitter
Choix ? 1
```

```
Votre appréciation ...
(M)eilleur
(R)ecommandé
(P)as recommandé
(N)SP
... pour 2 joueur(s) ? n
```

... pour 3 joueur(s) ? **r**
 ... pour 4 joueur(s) ? **m**

MENU :

1. Voter
2. Consulter les résultats
3. Quitter

Choix ? **1**

Votre appréciation ...

(M)eilleur

(R)ecommandé

(P)as recommandé

(N)SP

... pour 2 joueur(s) ? **n**

... pour 3 joueur(s) ? **m**

... pour 4 joueur(s) ? **r**

MENU :

1. Voter
2. Consulter les résultats
3. Quitter

Choix ? **2**

# joueurs	Meilleur	Recommandé	Pas recommandé	# votes
2	- (0)	- (0)	- (0)	0
3	50,0% (2)	50,0% (2)	0,0% (0)	4
4	66,7% (2)	33,3% (1)	0,0% (0)	3

MENU :

1. Voter
2. Consulter les résultats
3. Quitter

Choix ? **3**

Fin du programme.